meshService Mesh

Suresh Podeti

8 min read · Nov 6, 2023

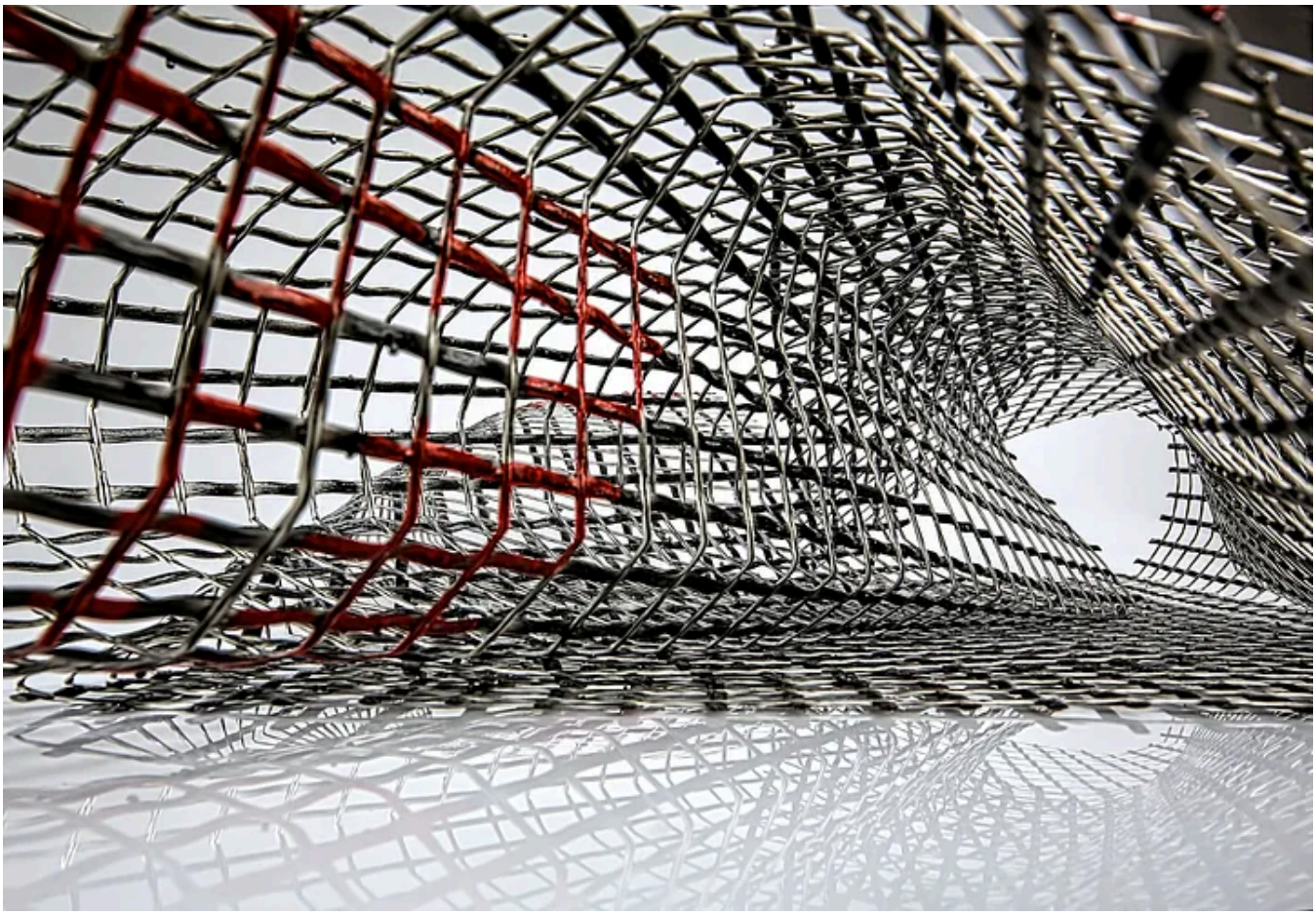

Photo by Ricardo Gomez Angel on Unsplash

Introduction

A service mesh is a dedicated infrastructure layer that controls service-to-service communication over a network. This method enables separate parts of an application to communicate with each other.

How service mesh is different from load balancer

Service meshes provide more advanced and fine-grained control over microservices in distributed architectures. While load balancers focus on distributing traffic, service meshes provide additional features such as **service discovery, security, observability, traffic management, and failure recovery**.

Service meshes can make service-to-service communication fast, reliable and secure.

What about API Gateway

An organization may choose an API gateway, which handles protocol transactions, instead of a service mesh. However, developers must update the API gateway every time a microservice is added or removed because API gateway needs to be aware of the available services and their endpoints. Service mesh typically offers network management scalability and flexibility that **exceeds** the capabilities of traditional API gateways.

A service mesh architecture uses a **proxy** instance called a **sidecar** in whichever development paradigm is in use, typically containers and/or microservices. In a microservice application, a sidecar attaches to each service. In a container, the sidecar attaches to each application container, VM or container orchestration unit, such as a Kubernetes pod

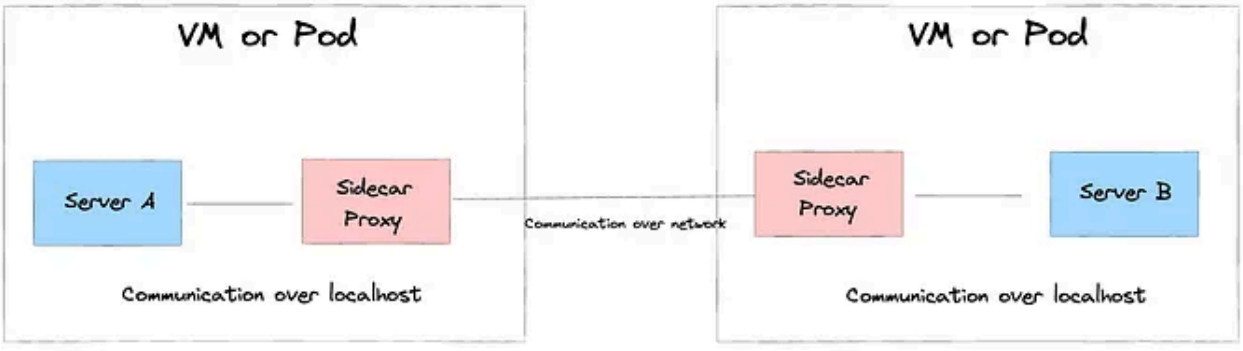

Fig 1.0: Deployment of server and proxy/sidecar

Is Service mesh only for micro-services?

While the service mesh pattern was designed to handle network connectivity between microservices, it can also be applied to other architectures (monolithic,

mini-services, serverless) wherever there are **multiple services communicating across a network.**

How?

Service mesh implementations usually have two main components:

- Data plane
- Control plane

The Data plane

The data plane is a network proxy replicated alongside each microservice (known as a sidecar), which manages all inbound and outbound network traffic on behalf of the microservice. **As part of this, it may perform service discovery, load balancing, security and reliability functions.**

Service instances, sidecars and their interactions make up what is called the data plane in a service mesh.

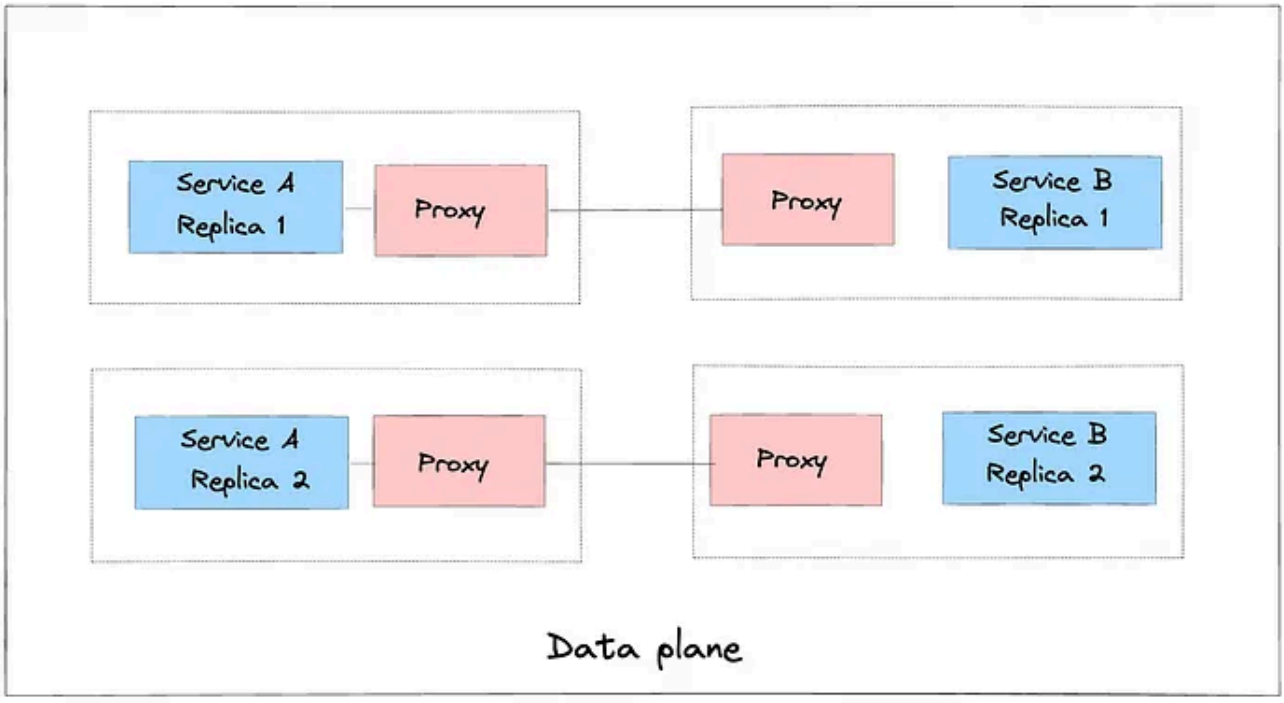

Fig 2.0: Dataplane in a service mesh

Many features provided by the service mesh work at the request level, making sidecars **Layer 7 proxies.** By operating at this layer, service meshes provide capabilities like intelligent load balancing based on observed latency, or provide sensible and consistent timeouts for requests.

The Control Plane

Proxies need to be configured. This is done through the control plane, which consists of several services that provide administrative functionality over the service mesh and provides an interface to configure the behavior of the data plane and for the proxies to coordinate their actions.

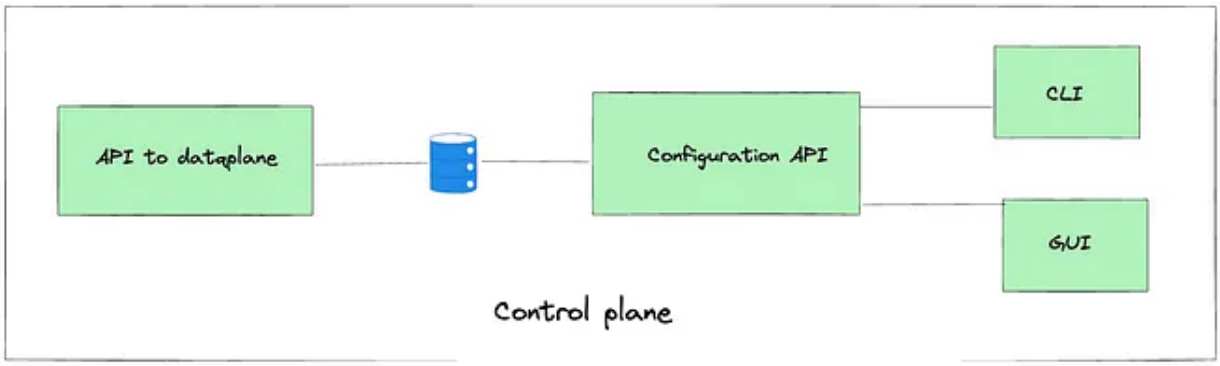

Fig 2.0: Control plane in service mesh

Operators interact with the service mesh through the control plane by using a CLI or API. For example, operators work through the control plane to define routing rules, create circuit breakers, or enforce access control.

Depending on the implementation, you can use the control plane to export observability data such as metrics, logs, and traces.

Benefits and Challenges

Distributed systems split applications into distinct services. While this architectural type has many advantages (like faster development time, making changes easier to implement, and ensuring resilience), it also introduces some challenges (like service discovery, efficient routing, and intelligent load balancing).

The service mesh layer allows applications to offload those implementations to the platform level. It also provides distinct advantages in the areas of observability, reliability, and security.

Observability

By default, sidecar proxies provide metrics, such as request latency and error counts. Proxies can automatically generate traces. Services forward these through necessary headers, thereby enhancing the visibility of requests flowing through the

system. Offloading observability concerns to the service mesh ensures consistent, useful observability.

Reliability

The service mesh also helps improve system reliability. By offloading fault tolerance concerns to a service mesh, services can focus on differentiating business logic. The service mesh can handle retrying requests transparently, without other services even being aware if dependencies are experiencing issues.

Service meshes also safeguard service reliability by enforcing a timeout on long-running requests. It can ensure services dont get overloaded by utilizing techniques like circuit breaking. Because the service mesh has a holistic system view, it can decide what techniques are necessary to maintain reliability.

Security

Service meshes can take on the responsibility of authenticating, controlling access to, and encrypting traffic between services. It can also help services mitigate issues like service impersonation, unauthorized access, or packet sniffing. These all take place at the platform level, without business applications intervening.

Service Mesh Challenges

A different layer called the control plane manages tasks such as creating instances, monitoring and implementing policies for network management and security. Control planes can connect to a CLI or a GUI interface for application management.

Why adopt a service mesh?

An application structured in a microservices architecture might comprise dozens or hundreds of services, all with their own instances that operate in a live environment. It’s a big challenge for developers to keep track of which components must interact, monitor their health and performance and make changes to a service or component if something goes wrong.

A service mesh enables developers to separate and manage service-to-service communications in a dedicated infrastructure layer. As the number of microservices involved with an application increases, so do the benefits of using a service mesh to manage and monitor them.

Key features of a service mesh

A service mesh framework typically provides many capabilities that make containerized and microservices communications more reliable, secure and

observable.

Reliability. Managing communications through sidecar proxies and the control plane improves efficiency and reliability of service requests, policies and configurations. Specific capabilities include load balancing and fault injection.

Observability. Service mesh frameworks can provide insights into the behavior and health of services. The control plane can collect and aggregate telemetry data from component interactions to determine service health, such as traffic and latency, distributed tracing and access logs. Third-party integration with tools, such as Prometheus, Elasticsearch and Grafana, enables further monitoring and visualization.

Security. Service mesh can automatically encrypt communications and distribute security policies, including authentication and authorization, from the network to the application and individual microservices. Centrally managing security policies through the control plane and sidecar proxies helps keep up with increasingly complex connections within and between distributed applications.

Service mesh benefits and drawbacks

A service mesh addresses some large issues with managing service-to-service communication, but not all. Some **advantages of a service mesh** are as follows:

- Simplifies communication between services in both microservices and containers.
- Easier to diagnose communication errors, because they would occur on their own infrastructure layer.
- Supports security features such as encryption, authentication and authorization.
- Allows for faster development, testing and deployment of an application.
- Sidecars placed next to a container cluster is effective in managing network services.

Some **downsides to service mesh** are as follows:

- Runtime instances increase through use of a service mesh.
- Each service call must first run through the sidecar proxy, which adds a step.

- Service meshes do not address integration with other services or systems, and routing type or transformation mapping.
- Network management complexity is abstracted and centralized, but not eliminated — someone must integrate service mesh into workflows and manage its configuration.

Despite all its benefits, the service mesh also comes with some caveats, which can present some challenges. When adopting a service mesh, you should consider the following.

- **Added complexity**: Adding a service mesh to a platform introduces another layer to your overall architecture. Regardless of which service mesh implementation you select, this will incur extra management costs. An extra set of services (such as the control plane) must be managed, and sidecar proxies must be configured and injected.
- **Resource consumption**: A sidecar proxy accompanies each application replica. These proxies consume resources (such as CPU and memory), increasing linearly with the number of application replicas.
- **Security loopholes**: Bugs or configuration mistakes at the service mesh layer can create security threats. For example, a wrong configuration can expose an internal service to the outside world.
- **Debugging**: An extra layer can make it harder to pinpoint issues. Traffic flowing through proxies adds extra network hops, which can obscure the root cause of some problems.

The threshold at which service mesh advantages exceed its disadvantages varies from organization to organization. When youre considering the adoption of a service mesh, it is crucial to know how they excel, what they can offer, and also when they can be counterproductive.

What Challenges Does Service Mesh Address?

The loosely coupled nature of services characterizes modern microservice architecture. But as the number of services and requests between them increases, the demands on the platform rise.

Service meshes can help to address these demands by providing solutions to common problems.

Service discovery

In microservice architectures, individual services must be able to find one another to communicate. Service meshes provide this functionality through a discovery layer. By registering services into the service mesh, other services can discover them by name and initiate communication.

Load balancing

Service meshes also allow independent scaling of services through easy and transparent load balancing between service replicas. In addition, they can provide load balancing algorithms ranging from simple round-robin balancing to more sophisticated algorithms, such as weighted or least requests.

Routing

Routing is another aspect where service meshes can benefit applications. Simpler architectures and processes dont require sophisticated routing. However, as platforms grow, this need emerges. Practices like A/B testing or canary deployments require that platforms can redirect requests to specific services (or service versions).

By using a service mesh, platforms can redirect users to specific service versions or direct requests based on specific headers or other criteria.

Reliability

Distributed systems need to handle fault-tolerance scenarios. When applications crash, it can cause cascading failures, unless a system can handle such scenarios gracefully. Service meshes improve system reliability by handling offloading techniques like circuit breaking.

Observability

By splitting services into microservices, engineers now must collect data from different sources to understand overall application behavior. Collecting data consistently from what could be hundreds of distributed services is no small task. In addition, that data needs to be correlated so that a system can be observable.

Service meshes handle this concern by collecting data about individual services, as well as the interactions between them, improving observability.

Inter-service Communication

Secure communication is another major concern in distributed systems. Splitting applications into multiple services increases the attack surface, potentially exposing applications to multiple attack vectors. Requests need to be authenticated, authorized, and encrypted. A service mesh can fulfill these requirements by employing techniques like role-based access control (RBAC), thereby managing secure communication at the platform level. It would be impractical and error-prone to address all of these concerns at the service level alone, and doing so would reduce much of the value of distributed systems.

Service meshes alleviate this complexity by offloading these concerns to the platform level. Services can focus on business logic while the service mesh takes care of the common system-level concerns.

The service mesh market

- Kong Mesh, HashiCorp Consul, F5 Nginx Service Mesh, Google Anthos, Istio, Linkerd.

Service Mesh | Software Architecture | Software Architect | System Design Interview | Distributed Systems

Edit profile

Written by Suresh Podeti

1.2K Followers

Aspiring Software Architect | SDE III (Golang) @JungleeGames| Ex-Byju's | Co-founder @Rupant Tech. | Ex-Synopsys | I.I.T Kharagpur | 7+ Years of Experience